**Python numpy tutorial**

NumPy is a Python library. NumPy is used for working with arrays.

NumPy is short form is "Numerical Python".

Arrays -1d ,2d ,3d … arrays

It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

**Why Use NumPy**?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up **to 50x faster** than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

## Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called **locality of reference** in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

## Import NumPy

## import numpy

## import numpy as np # as- change the name

## Checking NumPy Version:

print(np.\_\_version\_\_)

# NumPy Creating Arrays

|  |
| --- |
| arr = np.array([1, 2, 3, 4, 5])  print(arr)  print(type(arr)) |

Use a tuple to create a NumPy array:

## Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

## Shape of an Array :

The shape of an array is the number of elements in each dimension.

NumPy arrays have an attribute called **shape that returns a tuple** with each index having the number of corresponding elements.

# NumPy Array Indexing

## Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Get the first element from the following array:

## Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

## Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

|  |
| --- |
| # Access the third element of the second array of the first array:  import numpy as np  arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  print(arr[0, 1, 2])  o/p – 6 |

And this is why:

The first number represents the first dimension, which contains two arrays:  
[[1, 2, 3], [4, 5, 6]]  
and:  
[[7, 8, 9], [10, 11, 12]]  
Since we selected 0, we are left with the first array:  
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:  
[1, 2, 3]  
and:  
[4, 5, 6]  
Since we selected 1, we are left with the second array:  
[4, 5, 6]

The third number represents the third dimension, which contains three values:  
4  
5  
6  
Since we selected 2, we end up with the third value:  
6

## Negative Indexing

Use negative indexing to access an array from the end.

# NumPy Array Slicing

## Slicing arrays

**Slicing in python means taking elements from one given index to another given index**.

We pass slice instead of index like this: **[start:end].**

We can also define the step, like this: **[start:end:step].**

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

## Slicing 2-D Arrays

# NumPy Data Types

## Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

* i - integer
* b - boolean
* u - unsigned integer
* f - float
* c - complex float
* m - timedelta
* M - datetime
* O - object
* S - string
* U - unicode string
* V - fixed chunk of memory for other type ( void )

## Checking the Data Type of an Array

The NumPy array object has a property called **dtype that returns the data typ**e of the array:

prog

## Creating Arrays With a Defined Data Type

We use the **array()** function to create arrays, this function can take an optional argument**: dtype** that allows us to define the expected data type of the array elements: